TurboMouse: End-to-end Latency Compensation in Indirect Interaction

Axel Antoine
Université de Lille, France
axel.antoine@univ-lille.fr

Sylvain Malacria
Inria, France
sylvain.malacria@inria.fr

Géry Casiez
Université de Lille, France
gery.casiez@univ-lille.fr

To cite this work, please refer to the full publication [1] in *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI 2018)*

Abstract
End-to-end latency corresponds to the temporal difference between a user input and the corresponding output from a system. It has been shown to degrade user performance in both direct and indirect interaction. If it can be reduced to some extend, latency can also be compensated through software compensation by trying to predict the future position of the cursor based on previous positions, velocities and accelerations. In this paper, we propose a hybrid hardware and software prediction technique specifically designed for partially compensating end-to-end latency in indirect pointing. We combine a computer mouse with a high frequency accelerometer to predict the future location of the pointer using Euler based equations.

Author Keywords
computer mouse; accelerometer; end-to-end latency; prediction; jitter; performance.

ACM Classification Keywords
H.5.2 [User Interfaces]: Input devices and strategies

Introduction
End-to-end latency is defined as the total time required by a process, from the physical action of a user on an input device to the visual feedback on screen (i.e. sensing, transmission, system, applications, graphical computations and

Figure 1: Our setup comprises a Logitech G9 Laser Mouse connected via USB to the host computer with the MPU-9250 chip embedded inside. The MPU-9250 chip is connected to an Arduino Leonardo that is connected to the host computer via USB.

display refresh). End-to-end latency affects user experience with most interactive systems [9, 10, 2], from desktop computers (where latency is between 50 and 90 ms [2]) to touch-based devices (where latency is between 60 and 200 ms [10, 3]). And while modern input controllers (e.g. gaming controllers) have negligible input latency, they do not completely overcome the end-to-end latency [2, 9].

Latency significantly impacts performance in pointing and dragging tasks [7, 9, 11] and can be perceived from as low as 5-10 ms on a touch device [10]. Several methods have been explored to reduce or compensate for end-to-end latency, either controlling the entire graphical rendering pipeline [8], building homemade hardware devices and handling systems [10] or using *software compensation* by trying to predict the future positions based on previous positions, velocities and accelerations obtained by derivatives [6].

We present TurboMouse [1], *a hybrid hardware and software compensation* of end-to-end latency on desktop computers when operated using a computer-mouse, and we contribute a prediction technique specifically designed for indirect pointing. Our technique is based on the prediction of pointer position based on the velocity measured by the mouse and the acceleration from an accelerometer embedded in the mouse. Our assumption is that the hardware acceleration measured by the accelerometer is more precise and less noisy than a computed one from the velocity, which is captured discretely by the optical sensor of the mouse controller. Our system combines informations from the mouse and the accelerometer, and use Euler based equations to have a better estimation of the next position within a certain delay while reducing estimation errors and delays introduced by time derivative methods.

TurboMouse

Hardware

TurboMouse (Figure 1) relies on an optical laser computer mouse embedded with a low-cost high frequency accelerometer connected to an Arduino board. Both the mouse and Arduino board are connected to a host computer via USB.

Mouse controller: We used a conventional gaming USB Logitech G9 Laser mouse, whose retail price was of $99 USD. Its resolution can be configured from 200 CPI to 3200 CPI, and can send from 125 to 1000 HID reports per second. We use it in its default configuration, which runs at 1000 Hz with the resolution of 2000 CPI.

Accelerometer: Movement acceleration measures are provided by an InvenSense MPU-9250 chip, 9-axis motion tracking device embedded within the mouse controller, which is connected to an Arduino Leonardo board using I^2C communication system and positioned on top of the optical sensor. The MPU-9250 is able to run up to 4 kHz in a range of [-16g; +16g] with an initial tolerance of $\pm$ 80 mg. It also contains a gyroscope sensor, running up to 32 kHz and 2000 $^\circ/s$ and a magnetometer capable of running at 100 Hz. In this context, we only used the accelerometer component of the chip that we configured to run at 4 kHz in a range of [-2g ; +2g]. However, since the Leonardo is connected to host computer using USB protocol, we are limited to 1 kHz for sending raw HID reports. Therefore, we chose to read the register values at 1 kHz (i.e. read one value of four). We tried to compute an average each millisecond using the 4 values previously measured by the accelerometer, but this extra computation impacted the output rate of 1 kHz. The accelerometer of the MPU-9250 has a Zero-G initial calibration tolerance of $\pm$ 80 mg. To calibrate the accelerometer, we first record the acceleration values from the 3 axes at rest. From there, we compute a mean of each

axis acceleration, and we subtract the offset of each axis from the corresponding new input. To reduce noise, we also added filtering directly on the input acceleration and configured a 1€ filter [5] empirically tuned with 0.7 Hz *mincutoff* and 0.4 *beta* values for each axis.

Software
The mouse controller and Arduino board send their data to a host computer via USB that runs a dedicated software (written in C++ with the Qt Framework) to handle these signals and predict the next position of the mouse cursor.

Merging input data: Acceleration inputs are transmitted to the computer every 1ms using HIDAPI in a dedicated blocking-thread. Each event received is timestamped and buffered by our software. The Arduino also records and sends a timestamp to the host-computer. Mouse inputs are provided at 1 kHz by the mouse controller and captured by the *libpointing* library which allows us to get raw informations *(dx [counts], dy [counts], timestamp [ns], buttons)* at low level. We then associate each mouse input with the most recent acceleration values stored in the buffer.

Supporting transfer functions: Desktop systems commonly use non-linear *transfer functions* that dynamically adjust a *control-display gain* (G) that will impact cursor displacement [4] depending on the input velocity. Taking account of a transfer function *TF* for the input acceleration implies multiplying the inputs by this gain *G* as done for mouse inputs. However the operating system transferfunction makes it very hard to precisely know in real time the gain being applied on input. For that reason we bypass the system transfer function and use the transfer functions provided by *libpointing* to easily get the gain applied at a given time frame and exactly know the transfer function used.

Prediction: Our linear prediction technique retrieves the current cursor position $\mathbf{p_{current}}$ (in $meters$), the current cursor velocity $\mathbf{v}$ captured by the mouse (in m/s) and the current cursor acceleration $\mathbf{a}$ captured by the accelerometer ($m{\cdot}s^{-2}$) at high frequency (1 kHz). First, it computes the gain factor G based on the current transfer function TF and $\mathbf{v_{mouse}}$ (eq. 1). Then, it corrects the velocity $\mathbf{v}$ measured by the mouse optical sensor every *dt* with hardware acceleration $\mathbf{a}$ to obtain a more accurate velocity $\mathbf{v_{corrected}}$ (eq. 2). *dt* is computed as the timestamps difference between the current event and the past event.

$$G = \frac{TF(v_{mouse})}{v_{mouse}} \tag{1}$$

$$\mathbf{v_{corrected}} = \mathbf{v_{mouse}} * G + \mathbf{a} * G * dt \tag{2}$$

$$\mathbf{p_{predicted}} = \mathbf{p_{current}} + \mathbf{v_{corrected}} * comp + 0.5 * \mathbf{a} * G * comp^2 \tag{3}$$

From there, the predicted position $\mathbf{p_{predicted}}$ of the mouse can be computed within a future time interval (*comp*, for latency compensation) using the current position $\mathbf{p_{current}}$, the corrected velocity $\mathbf{v_{corrected}}$ and the acceleration $\mathbf{a}$ (eq. 3). We also smoothed the predicted positions with 1€ filter [5] empirically tuned with 12 Hz *mincutoff* and 0 *beta* values for each axis (x and y).

Conclusion

We introduced TurboMouse, a hybrid hardware and software latency compensation technique specifically designed for indirect interaction. TurboMouse combines the velocity measured by the mouse and the acceleration reported by an accelerometer embedded in the mouse to predict cursor's position using Euler based equations.

REFERENCES

1. Axel Antoine, Sylvain Malacria, and Géry Casiez. 2018. Using High Frequency Accelerometer and Mouse to Compensate for End-to-end Latency in Indirect Interaction. In *Proc. CHI'18*. ACM. DOI: http://dx.doi.org/10.1145/3173574.3174183
2. Géry Casiez, Stéphane Conversy, Matthieu Falce, Stéphane Huot, and Nicolas Roussel. 2015. Looking Through the Eye of the Mouse: A Simple Method for Measuring End-to-end Latency Using an Optical Mouse. In *Proc. UIST '15*. ACM, 629–636. DOI: http://dx.doi.org/10.1145/2807442.2807454
3. Géry Casiez, Thomas Pietrzak, Damien Marchal, Sébastien Poulmane, Mathieu Falce, and Nicolas Roussel. 2017. Characterizing Latency in Touch and Button-Equipped Interactive Systems. In *Proc. UIST '17*. ACM, 9. DOI: http://dx.doi.org/10.1145/3126594.3126606
4. Géry Casiez and Nicolas Roussel. 2011. No More Bricolage! Methods and Tools to Characterize, Replicate and Compare Pointing Transfer Functions. In *Proc. UIST '11*. ACM, 603–614. DOI: http://dx.doi.org/10.1145/2047196.2047276
5. Géry Casiez, Nicolas Roussel, and Daniel Vogel. 2012. 1€ Filter: A Simple Speed-based Low-pass Filter for Noisy Input in Interactive Systems. In *Proc. CHI'12*. ACM, 2527–2530. DOI: http://dx.doi.org/10.1145/2207676.2208639
6. Elie Cattan, Amélie Rochet-Capellan, Pascal Perrier, and François Bérard. 2015. Reducing Latency with a Continuous Prediction: Effects on Users' Performance in Direct-Touch Target Acquisitions. In *Proc. ITS'15*. ACM, New York, NY, USA, 205–214. DOI: http://dx.doi.org/10.1145/2817721.2817736
7. S. Friston, P. KarlstrÃűm, and A. Steed. 2016a. The Effects of Low Latency on Pointing and Steering Tasks. *IEEE Transactions on Visualization and Computer Graphics* 22, 5 (May 2016), 1605–1615. DOI: http://dx.doi.org/10.1109/TVCG.2015.2446467
8. Sebastian Friston, Anthony Steed, Simon Tilbury, and Georgi Gaydadjiev. 2016b. Construction and Evaluation of an Ultra Low Latency Frameless Renderer for VR. *IEEE Transactions on Visualization and Computer Graphics* 22, 4 (April 2016), 1377–1386. DOI: http://dx.doi.org/10.1109/TVCG.2016.2518079
9. I. Scott MacKenzie and Colin Ware. 1993. Lag As a Determinant of Human Performance in Interactive Systems. In *Proc. CHI '93 (CHI '93)*. ACM, 488–493. DOI:http://dx.doi.org/10.1145/169059.169431
10. Albert Ng, Julian Lepinski, Daniel Wigdor, Steven Sanders, and Paul Dietz. 2012. Designing for Low-latency Direct-touch Input. In *Proc. UIST '12 (UIST '12)*. ACM, 453–464. DOI: http://dx.doi.org/10.1145/2380116.2380174
11. Andriy Pavlovych and Wolfgang Stuerzlinger. 2009. The Tradeoff Between Spatial Jitter and Latency in Pointing Tasks. In *Proc. EICS '09*. ACM, 187–196. DOI:http://dx.doi.org/10.1145/1570433.1570469